

Thomas Decomposition of Algebraic and Differential Systems

Thomas Bächler¹ `thomas@momo.math.rwth-aachen.de`, Vladimir Gerdt² `gerdt@jinr.ru`,
 Markus Lange-Hegermann¹ `markus@momo.math.rwth-aachen.de` and Daniel Robertz¹
`daniel@momo.math.rwth-aachen.de`

¹ Lehrstuhl B für Mathematik, RWTH-Aachen University, Germany

² Joint Institute for Nuclear Research, Dubna, Russia

Abstract. In this paper we consider disjoint decomposition of algebraic and non-linear partial differential systems of equations and inequations into so-called simple subsystems. We exploit THOMAS decomposition ideas and develop them into a new algorithm. For algebraic systems simplicity means triangularity, squarefreeness and non-vanishing initials. For differential systems the algorithm provides not only algebraic simplicity but also involutivity. The algorithm has been implemented in MAPLE.

1 Introduction

Nowadays, triangular decomposition algorithms, which go back to the characteristic set method by Ritt [Rit50] and Wu [Wu00], and software implementing them have become powerful tools for investigating and solving systems of multivariate polynomial equations. In many cases these methods are computationally more efficient than those based on construction of GRÖBNER bases. As an example of such problems one can indicate BOOLEAN polynomial systems arising in cryptanalysis of stream ciphers. For those systems triangular decomposition algorithms based on the characteristic set method revealed their superiority over the best modern algorithms for the construction of GRÖBNER bases [sGH09].

For terminology, literature, definitions and basic proofs on triangular-decomposition algorithms for polynomial and differential-polynomial systems we refer to the excellent tutorial papers [Hub03a, Hub03b] and to the bibliographical references therein.

Among numerous triangular decompositions the THOMAS one stands by itself. It was suggested by the American mathematician J.M.Thomas in his books [Tho37, Tho62] and decomposes a finite system of polynomial equations and inequations into finitely many triangular subsystems that he called *simple*. Unlike other decomposition algorithms it yields a *disjoint* zero decomposition, that is, it decomposes the affine variety or quasi-affine variety defined by the input into a finite number of disjoint quasi-affine varieties determined by the output simple systems. Every simple system is a regular chain.

During his research on triangular decomposition, Thomas was motivated by the RIQUIER-JANET theory [Riq10, Jan29], extending it to non-linear systems of partial differential equations. For this purpose he developed a theory of (THOMAS) monomials, which generate the involutive monomial division called THOMAS division in [GB98a]. He gave a recipe for decomposing a non-linear differential system into algebraically simple and passive subsystems [Tho37].

Differential THOMAS decomposition differs noticeably from that computed by the famous ROSENFELD-GRÖBNER algorithm [BLOP09, BLOP95] which forms a basis of the `difalg` and `BLAD` libraries [BH04, Bou09] as well as from other differential decompositions (e.g. [BKRM01]). We found that `difalg` and `BLAD` are optimized and well-suited for ordinary differential equations. However, all other known methods give a zero decomposition which, unlike that in THOMAS decomposition, is not necessarily disjoint.

A first implementation of the THOMAS decomposition was done by Teresa Gómez-Díaz in AXIOM under the name “dynamic constructible closure” which later turned out to be the same as the THOMAS decomposition [Del00]. Wang later designed and implemented an algorithm constructing the THOMAS decomposition [Wan98, Wan01, LW99]. For polynomial and ordinary differential

systems Wang's algorithm was implemented by himself in MAPLE [Wan04] as part of the software package `epsilon` [Wan03], which also contains implementations of a number of other triangular decomposition algorithms. A modified algorithmic version of the THOMAS decomposition was considered in [Ger08] with its link to the theory of involutive bases [GB98a, Ger05, Ger99]. The latter theory together with some extensions is presented in detail in the recent book [Sei10].

In the given paper we present a new algorithmic version of the THOMAS decomposition for polynomial and (partial) differential systems. In the differential case the output subsystems are JANET involutive in accordance to the involutivity criterion from [Ger08], and hence they are coherent. Moreover, for every output subsystem the set of its equations is a minimal JANET basis of the radical differential ideal generated by this set. The algorithm has been implemented in MAPLE for both the algebraic and differential case. For a linear differential system it constructs a JANET basis of the corresponding differential ideal and for this case works similarly to the MAPLE package `Janet` (cf. [BCG⁺03]).

This paper is organized as follows. In §2 we sketch the algebraic part of our algorithm for the THOMAS decomposition with its main objects defined in §2.1. The algorithm itself together with its subalgorithms is considered in §2.2. Decomposition of differential systems is described in §3. Here we briefly introduce some basic notions and concepts from differential algebra (§3.1) and from the theory of involutive bases specific to JANET division (§3.2) together with one of the two extra subalgorithms that extend the algebraic decomposition to the differential one. The second such subalgorithm is considered in §3.3 along with the definition of differential simple systems. Subsection §3.4 contains a description of the differential THOMAS decomposition algorithm. Some implementation issues are discussed in §4, where we also demonstrate the MAPLE implementation for the differential decomposition using the example of a system related to control theory.

We omit the proofs for compactness. They will be published elsewhere.

2 Algebraic Systems

The algebraic THOMAS decomposition deals with systems of polynomial equations and inequations. This section introduces the concepts of simple systems and disjoint decompositions based on properties of the set of solutions of a system. A pseudo reduction procedure and several splitting algorithms on the basis of polynomial remainder sequences are introduced as tools for the main algorithm, which is presented at the end of the section.

2.1 Preliminaries

Let F be a computable field of characteristic 0 and $R := F[x_1, \dots, x_n]$ the polynomial ring in n variables. A total order $<$ on the indeterminates of R is called a **ranking**. The notation $R = F[x_1, \dots, x_n]$ shall implicitly define the ranking $x_1 < \dots < x_n$. The indeterminate x is called **leader** of $p \in R$ if x is the $<$ -largest variable occurring in p and we write $\text{ld}(p) = x$. If $p \in F$, we define $\text{ld}(p) = 1$ and $1 < x$ for all indeterminates x . The degree of p in $\text{ld}(p)$ is called **rank** of p and the leading coefficient $\text{init}(p) \in F[y \mid y < \text{ld}(p)]$ of $\text{ld}(p)^{\text{rank}(p)}$ in p is called **initial** of p .

For $\mathbf{a} \in \overline{F}^n$, where \overline{F} denotes the algebraic closure of F , define the following evaluation homomorphisms:

$$\begin{aligned} \phi_{\mathbf{a}} : F[x_1, \dots, x_n] &\rightarrow \overline{F} : x_i \mapsto a_i \\ \phi_{<_{x_k}, \mathbf{a}} : F[x_1, \dots, x_n] &\rightarrow \overline{F}[x_k, \dots, x_n] : \begin{cases} x_i \mapsto a_i, & i < k \\ x_i \mapsto x_i, & \text{otherwise} \end{cases} \end{aligned}$$

For a polynomial $p \in R$, the symbols $p_ =$ and p_{\neq} shall denote the equation $p = 0$ and inequation $p \neq 0$, respectively. A finite set of equations and inequations is called an **(algebraic) system** over R . Abusing notation, we sometimes treat $p_ =$ or p_{\neq} as the underlying polynomial p . A **solution** of a system S is a tuple $\mathbf{a} \in \overline{F}^n$ such that $\phi_{\mathbf{a}}(p) = 0$ for all equations $p_ = \in S$ and $\phi_{\mathbf{a}}(p) \neq 0$ for all inequations $p_{\neq} \in S$. The set of all solutions of S is denoted by $\text{Sol}(S)$.

Define $S_x := \{p \in S \mid \text{ld}(p) = x\}$. In a situation where it is clear that $|S_x| = 1$, we also use S_x to denote the unique element of S_x . The subset $S_{<x} := \{p \in S \mid \text{ld}(p) < x\}$ can be considered a system over $F[y \mid y < x]$. Furthermore, the sets of all equations $p_+ \in S$ and all inequations $p_- \in S$ are denoted by $S^=$ and S^\neq , respectively.

The general idea of the THOMAS methods is to use the homomorphism $\phi_{<x, \mathbf{a}}$ to treat each polynomial $p \in S_x$ as the *univariate* polynomial $\phi_{<x, \mathbf{a}}(p) \in \overline{F}[x]$ for all $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$ *simultaneously*. This idea forms the basis of our central object, the **simple system**:

Definition 2.1 (Simple Systems). Let S be a system.

1. S is **triangular** if $|S_{x_i}| \leq 1 \forall 1 \leq i \leq n$ and $S \cap \{c_+, c_- \mid c \in F\} = \emptyset$.
2. S has **non-vanishing initials** if $\phi_{\mathbf{a}}(\text{init}(p)) \neq 0 \forall \mathbf{a} \in \mathfrak{Sol}(S_{<x_i})$ and $p \in S_{x_i}$ for $1 \leq i \leq n$.
3. S is **square-free** if the univariate polynomial $\phi_{<x_i, \mathbf{a}}(p) \in \overline{F}[x_i]$ is square-free $\forall \mathbf{a} \in \mathfrak{Sol}(S_{<x_i})$ and $p \in S_{x_i}$ for $1 \leq i \leq n$.
4. S is called **simple** if it is *triangular*, has *non-vanishing initials* and is *square-free*.

Although all required properties are characterized via solutions of lower-ranking equations and inequations, the THOMAS decomposition algorithm does not calculate solutions of polynomials. Instead, it uses polynomial equations and inequations to *partition* the set of solutions of the lower-ranking system to ensure the above properties.

Remark 2.2. Simplicity of a system guarantees the existence of solutions: If $\mathbf{b} \in \mathfrak{Sol}(S_{<x})$ and S_x is not empty, then $\phi_{<x, \mathbf{b}}(S_x)$ is a univariate polynomial with exactly $\text{rank}(S_x)$ *distinct* roots. When extending \mathbf{b} to a solution (\mathbf{b}, a) of $S_{\leq x}$, for an equation in S_x there are $\text{rank}(S_x)$ choices for a , whereas for an inequation or empty S_x all but finitely many $a \in \overline{F}$ give an extension.

To transform a system into a simple system, it is in general necessary to partition the set of solutions. Instead of an equivalent simple system, this leads to a so-called decomposition into simple systems.

Definition 2.3. A family $(S_i)_{i=1}^m$ is called **decomposition** of S if $\mathfrak{Sol}(S) = \bigcup_{i=1}^m \mathfrak{Sol}(S_i)$. A decomposition is called **disjoint** if $\mathfrak{Sol}(S_i) \cap \mathfrak{Sol}(S_j) = \emptyset \forall i \neq j$. A *disjoint* decomposition of a system into *simple systems* is called **(algebraic) THOMAS decomposition**.

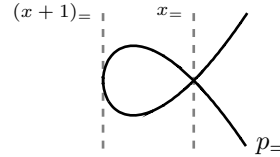
For any algebraic system S , there exists a THOMAS decomposition (cf. [Tho37], [Tho62], [Wan98]). The algorithm presented in the following section provides another proof of this fact. First, we give an easy example of a THOMAS decomposition.

Example 2.4. Consider the equation

$$p = y^2 - x^3 - x^2.$$

A THOMAS decomposition of $\{p_+\}$ is given by:

$$(\{(y^2 - x^3 - x^2)_+, (x \cdot (x+1))_\neq\}, \{y_+, (x \cdot (x+1))_+\})$$



2.2 Decomposition Algorithms

Our version of the decomposition algorithm in each round treats one system, potentially splitting it into several subsystems. For this purpose, one polynomial is chosen from a list of polynomials to be processed. This polynomial is pseudo-reduced modulo the system and afterwards combined with the polynomial in the system having the same leader. To ensure that all polynomials are square-free and their initials do not vanish, the system may be split into several ones by initials of polynomials or subresultants.

From now on, a system S is presented as a pair of sets (S_T, S_Q) , where S_T represents a candidate for a simple system while S_Q is the queue of elements to be processed. S_T is always triangular and $(S_T)_x$ denotes the unique equation or inequation of leader x in S_T , if any. S_T also fulfills a weaker

form of the other two simplicity conditions, i.e., for any solution \mathbf{a} of $(S_T)_{<x} \cup (S_Q)_{<x}$, we have $\phi_{\mathbf{a}}(\text{init}((S_T)_x)) \neq 0$ and $\phi_{<x,\mathbf{a}}((S_T)_x)$ is square-free.

From now on, let **prem** be a **pseudo remainder algorithm**³ in R and **pquo** the corresponding **pseudo quotient algorithm**, i.e., for p and q with $\text{ld}(p) = \text{ld}(q) = x$

$$m \cdot p = \text{pquo}(p, q, x) \cdot q + \text{prem}(p, q, x) \quad (1)$$

where $\deg_x(q) > \deg_x(\text{prem}(p, q, x))$ and $m \in R \setminus \{0\}$ with $\text{ld}(m) < x$ and $m \mid \text{init}(q)^k$ for some $k \in \mathbb{Z}_{\geq 0}$. Note that if the initials of p and q are non-zero, the initial of $\text{pquo}(p, q, x)$ is also non-zero. Equation (1) only allows us to replace p with $\text{prem}(p, q, x)$ if m does not vanish on any solution. The below Algorithm (2.5) and Remark (2.6) require the last property, which, by definition, holds in simple systems.

The following algorithm employs pseudo remainders and the triangular structure to reduce a polynomial modulo S_T :

Algorithm 2.5 (Reduce).

Input: A system S , a polynomial $p \in R$

Output: A polynomial q with $\phi_{\mathbf{a}}(p) = 0$ if and only if $\phi_{\mathbf{a}}(q) = 0$ for each $\mathbf{a} \in \mathfrak{Sol}(S)$.

Algorithm:

```

1:  $x \leftarrow \text{ld}(p)$ ;  $q \leftarrow p$ 
2: while  $x > 1$  and  $(S_T)_x$  is an equation and  $\text{rank}(q) \geq \text{rank}((S_T)_x)$  do
3:    $q \leftarrow \text{prem}(q, (S_T)_x, x)$ 
4:    $x \leftarrow \text{ld}(q)$ 
5: end while
6: if  $x > 1$  and  $\text{Reduce}(S, \text{init}(q)) = 0$  then
7:   return  $\text{Reduce}(S, q - \text{init}(q)x^{\text{rank}(q)})$ 
8: else
9:   return  $q$ 
10: end if
```

A polynomial p is called **reduced modulo S_T** if $\text{Reduce}(S, p) = p$. A polynomial p **reduces to q modulo S_T** if $\text{Reduce}(S, p) = q$.

The result of the Reduce algorithm does not need to be a canonical normal form. It only needs to detect polynomials that vanish on all solutions of a system:

Remark 2.6. Let $p \in R$ with $\text{ld}(p) = x$. $\text{Reduce}(S, p) = 0$ implies $\phi_{\mathbf{a}}(p) = 0 \forall \mathbf{a} \in \mathfrak{Sol}(S_{\leq x})$.

The converse of this remark only holds if $(S_Q)_{\leq x} = \emptyset$, i.e., $(S_T)_{\leq x}$ is simple. If it is not simple, but $\text{ld}(p) = x$ and $(S_Q)_{<x} = \emptyset$ hold, we still have some information. In particular, $\text{Reduce}(S, p) \neq 0$ implies that either $\mathfrak{Sol}(S_{<x})$ is empty or there exists $\mathbf{a} \in \mathfrak{Sol}(S_{<x} \cup \{(S_T)_x\})$ such that $\phi_{\mathbf{a}}(p) \neq 0$.

We now direct our attention to the methods we use to produce disjoint decompositions. Since $(S \cup \{p \neq\}, S \cup \{p =\})$ is a disjoint decomposition of S , we will use the following one-line subalgorithm as the basis of all the splitting algorithms described below.

Algorithm 2.7 (Split). *Input:* A system S , a polynomial $p \in R$

Output: The disjoint decomposition $(S \cup \{p \neq\}, S \cup \{p =\})$ of S .

Algorithm:

```

1: return  $((S_T, S_Q \cup \{p \neq\}), (S_T, S_Q \cup \{p =\}))$ 
```

The output of the following splitting algorithms is not yet a disjoint decomposition of the input. However, the main algorithm **Decompose** will use this output to construct a disjoint decomposition. We single out these algorithms to make the main algorithm more compact and readable. For details we refer to the input and output specifications of the algorithms in question.

The algorithm **InitSplit** ensures that in one of the returned systems the property 2 in Definition (2.1) holds for the input polynomial. In the other system the initial of that polynomial vanishes.

³ In our context **prem** does not necessarily have to be the classical pseudo remainder, but any sparse pseudo remainder with property (1) will suffice.

Algorithm 2.8 (InitSplit). *Input:* A system S , an equation or inequation q with $\text{ld}(q) = x$.
Output: Two systems S_1 and S_2 , where $(S_1 \cup \{q\}, S_2)$ is a disjoint decomposition of $S \cup \{q\}$.
 Moreover, $\phi_{\mathbf{a}}(\text{init}(q)) \neq 0$ holds for all $\mathbf{a} \in \mathfrak{Sol}(S_1)$ and $\phi_{\mathbf{a}}(\text{init}(q)) = 0$ for all $\mathbf{a} \in \mathfrak{Sol}(S_2)$.
Algorithm:

```

1:  $(S_1, S_2) \leftarrow \text{Split}(S, \text{init}(q))$ 
2: if  $q$  is an equation then
3:    $(S_2)_Q \leftarrow (S_2)_Q \cup \{(q - \text{init}(q)x^{\text{rank}(q)})_{=}\}$ 
4: else if  $q$  is an inequation then
5:    $(S_2)_Q \leftarrow (S_2)_Q \cup \{(q - \text{init}(q)x^{\text{rank}(q)})_{\neq}\}$ 
6: end if
7: return  $(S_1, S_2)$ 
    
```

In Definition (2.1) we view a multivariate polynomial p as the univariate polynomial $\phi_{<\text{ld}(p), \mathbf{a}}(p)$. For ensuring triangularity and square-freeness, we often compute the gcd of two polynomials, which generally depends on the inserted value \mathbf{a} . Subresultants provide a generalization of the EUCLIDEAN algorithm useful in our context and their initials distinguish the cases of different degrees of gcds.

Definition 2.9. Let $p, q \in R$ with $\text{ld}(p) = \text{ld}(q) = x$, $\deg_x(p) = d_p > \deg_x(q) = d_q$. We denote by $\text{PRS}(p, q, x)$ the **subresultant polynomial remainder sequence** (see [Hab48], [Mis93, Chap. 7], [Yap00, Chap. 3]) of p and q w.r.t. x , and by $\text{PRS}_i(p, q, x)$, $i < d_q$ the regular polynomial of degree i in $\text{PRS}(p, q, x)$ if it exists, or 0 otherwise. Furthermore, $\text{PRS}_{d_p}(p, q, x) := p$, $\text{PRS}_{d_q}(p, q, x) := q$ and $\text{PRS}_i(p, q, x) := 0$, $d_q < i < d_p$.

Define $\text{res}_i(p, q, x) := \text{init}(\text{PRS}_i(p, q, x))$ for $0 < i < d_p$, whereas $\text{res}_{d_p}(p, q, x) := 1$ and $\text{res}_0(p, q, x) := \text{PRS}_0(p, q, x)$. Note that $\text{res}(p, q, x) := \text{res}_0(p, q, x)$ is the usual resultant.

Definition 2.10. Let S be a system and $p_1, p_2 \in R$ with $\text{ld}(p_1) = \text{ld}(p_2) = x$. If $|\mathfrak{Sol}(S_{<x})| > 0$, we call

$$i := \min \{i \in \mathbb{Z}_{\geq 0} \mid \exists \mathbf{a} \in \mathfrak{Sol}(S_{<x}) \text{ such that } \deg_x(\gcd(\phi_{<x, \mathbf{a}}(p_1), \phi_{<x, \mathbf{a}}(p_2))) = i\}$$

the **fiber cardinality** of p_1 and p_2 w.r.t. S . Moreover, if $(S_Q)_{<x}^{\neq} = \emptyset$, then

$$i' := \min \{i \in \mathbb{Z}_{\geq 0} \mid \text{Reduce}(\text{res}_j(p_1, p_2, x), S_T) = 0 \ \forall \ j < i \text{ and } \text{Reduce}(\text{res}_i(p_1, p_2, x), S_T) \neq 0\}$$

is the **quasi fiber cardinality** of p_1 and p_2 w.r.t. S . A disjoint decomposition (S_1, S_2) of S such that

1. $\deg_x(\gcd(\phi_{<x, \mathbf{a}}(p_1), \phi_{<x, \mathbf{a}}(p_2))) = i \ \forall \ \mathbf{a} \in \mathfrak{Sol}((S_1)_{<x})$
2. $\deg_x(\gcd(\phi_{<x, \mathbf{a}}(p_1), \phi_{<x, \mathbf{a}}(p_2))) > i \ \forall \ \mathbf{a} \in \mathfrak{Sol}((S_2)_{<x})$

is called the i -th **fibration split** of p_1 and p_2 w.r.t. S . A polynomial $r \in R$ with $\text{ld}(r) = x$ such that $\deg_x(r) = i$ and

$$\phi_{<x, \mathbf{a}}(r) \sim \gcd(\phi_{<x, \mathbf{a}}(p_1), \phi_{<x, \mathbf{a}}(p_2)) \ \forall \ \mathbf{a} \in \mathfrak{Sol}((S_1)_{<x})$$

is called the i -th **conditional greatest common divisor** of p_1 and p_2 w.r.t. S , where $p \sim q$ if and only if $p \in \overline{K}^* q$. Furthermore, $q \in R$ with $\text{ld}(q) = x$ and $\deg_x(q) = \deg_x(p_1) - i$ such that

$$\phi_{<x, \mathbf{a}}(q) \sim \frac{\phi_{<x, \mathbf{a}}(p_1)}{\gcd(\phi_{<x, \mathbf{a}}(p_1), \phi_{<x, \mathbf{a}}(p_2))} \ \forall \ \mathbf{a} \in \mathfrak{Sol}((S_1)_{<x})$$

is called the i -th **conditional quotient** of p_1 by p_2 w.r.t. S . By replacing $\phi_{<x, \mathbf{a}}(p_2)$ in the above definition with $\frac{\partial}{\partial x}(\phi_{<x, \mathbf{a}}(p_1))$, we get an i -th **square-free split** and i -th **conditional square-free part** of p_1 w.r.t. S .

The fiber cardinality is often not immediately available, as we may be unable to take inequations into account. However, we can use all information contained in the equations using reduction, if all equations are contained in S_T . Thus we require $(S_Q)_{<x}^{\overline{=}} = \emptyset$ before doing any reduction.

In this situation, the quasi fiber cardinality is easy to calculate and in many cases will be identical to the fiber cardinality. Furthermore, if we consider the system S_2 from an i -th fibration split of some polynomials for a system S and ensure that $((S_2)_Q)_{<x}^{\overline{=}} = \emptyset$, then the quasi fiber cardinality of the same polynomials for S_2 will be $i + 1$. Therefore and due to the following lemma, the quasi fiber cardinality is good enough for our purposes.

Lemma 2.11. Let $|\mathfrak{Sol}(S_{<x})| > 0$ and $(S_Q)_{<x}^{\overline{=}} = \emptyset$. For p_1, p_2 as in Definition (2.10) with $\phi_{\mathbf{a}}(\text{init}(p_1)) \neq 0 \forall \mathbf{a} \in \mathfrak{Sol}(S_{<x})$ and $\text{rank}(p_1) > \text{rank}(p_2)$, let i be the fiber cardinality of p_1 and p_2 w.r.t. S and i' the corresponding quasi fiber cardinality. Then

$$i' \leq i$$

where the equality holds if and only if $|\mathfrak{Sol}(S_{<x} \cup \{\text{res}_{i'}(p_1, p_2, x)_{\neq}\})| > 0$.

Corollary 2.12. Let $|\mathfrak{Sol}(S_{<x})| > 0$ and $(S_Q)_{<x}^{\overline{=}} = \emptyset$. For polynomials p_1, p_2 as in Definition (2.10) with $\phi_{\mathbf{a}}(\text{init}(p_1)) \neq 0$ and $\phi_{\mathbf{a}}(\text{init}(p_2)) \neq 0 \forall \mathbf{a} \in \mathfrak{Sol}(S_{<x})$, let i be the fiber cardinality of p_1 and p_2 w.r.t. S and i' the quasi fiber cardinality of p_1 and $\text{prem}(p_2, p_1, x)$ w.r.t. S . Then

$$i' \leq i$$

with equality if and only if $|\mathfrak{Sol}(S_{<x} \cup \{\text{res}_{i'}(p_1, \text{prem}(p_2, p_1, x), x)_{\neq}\})| > 0$.

The following algorithm calculates the quasi fiber cardinality of two polynomials. It is used as the basis for all algorithms that calculate a greatest common divisor or a least common multiple.

Algorithm 2.13 (ResSplit). *Input:* A system S with $(S_Q)_{<x}^{\overline{=}} = \emptyset$, two polynomials $p, q \in R$ with $\text{ld}(p) = \text{ld}(q) = x$, $\text{rank}(p) > \text{rank}(q)$ and $\phi_{\mathbf{a}}(\text{init}(p)) \neq 0$ for all $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$.

Output: The quasi fiber cardinality i of p and q w.r.t. S and an i -th fibration split (S_1, S_2) of p and q w.r.t. S .

Algorithm:

- 1: $i \leftarrow \min\{i \in \mathbb{Z}_{\geq 0} \mid \text{Reduce}(\text{res}_j(p, q, x), S_T) = 0 \forall j < i \text{ and } \text{Reduce}(\text{res}_i(p, q, x), S_T) \neq 0\}$
- 2: **return** $(i, S_1, S_2) := (i, \text{Split}(S, \text{res}_i(p, q, x)))$

Similarly to the `InitSplit` algorithm (2.8), the following algorithm does not return a disjoint decomposition, but `Decompose` uses it to construct one.

Algorithm 2.14 (ResSplitGCD). *Input:* A system S with $(S_Q)_{<x}^{\overline{=}} = \emptyset$, where $(S_T)_x$ is an equation, and an equation $q_{=}$ with $\text{ld}(q) = x$. Furthermore $\text{rank}(q) < \text{rank}((S_T)_x)$.

Output: Two systems S_1 and S_2 and an equation $\tilde{q}_{=}$ such that:

- a) $S_2 = \widetilde{S_2} \cup \{q\}$ where $(S_1, \widetilde{S_2})$ is an i -th fibration split of $(S_T)_x$ and q w.r.t. S
- b) \tilde{q} is an i -th conditional gcd of $(S_T)_x$ and q w.r.t. S .

where i is the quasi fiber cardinality of p and q w.r.t. S .

Algorithm:

- 1: $(i, S_1, S_2) \leftarrow \text{ResSplit}(S, (S_T)_x, q)$
- 2: $(S_2)_Q \leftarrow (S_2)_Q \cup \{q\}$
- 3: **return** $S_1, S_2, \text{PRS}_i((S_T)_x, q, x)_{=}$

The following algorithm is similar, but instead of the gcd, it returns the first input polynomial divided by the gcd. It is used to assimilate an inequation into a system where there already is an equation with the same leader, or to calculate the least common multiple of two inequations.

Algorithm 2.15 (ResSplitDivide). *Input:* A system S with $(S_Q)_{<x}^{\equiv} = \emptyset$ and two polynomials p, q with $\text{ld}(p) = \text{ld}(q) = x$ and $\phi_{\mathbf{a}}(\text{init}(p)) \neq 0$ for all $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$. Furthermore, if $\text{rank}(p) \leq \text{rank}(q)$ then $\phi_{\mathbf{a}}(\text{init}(q)) \neq 0$.

Output: Two systems S_1 and S_2 and a polynomial \tilde{p} such that:

- a) $S_2 = \widetilde{S_2} \cup \{q\}$ where $(S_1, \widetilde{S_2})$ is an i -th fibration split p and q' w.r.t. S
- b) \tilde{p} is an i -th conditional quotient of p by q' w.r.t. S

where i is the quasi fiber cardinality of p and q' w.r.t. S , with $q' = q$ for $\text{rank}(p) > \text{rank}(q)$ and $q' = \text{prem}(q, p, x)$ otherwise.

Algorithm:

```

1: if  $\text{rank}(p) \leq \text{rank}(q)$  then
2:   return ResSplitDivide( $S, p, \text{prem}(q, p, x)$ )
3: else
4:    $(i, S_1, S_2) \leftarrow \text{ResSplit}(S, p, q)$ 
5:   if  $i > 0$  then
6:      $\tilde{p} \leftarrow \text{pquo}(p, \text{PRS}_i(p, \text{prem}(q, p, x), x), x)$ 
7:   else
8:      $\tilde{p} \leftarrow p$ 
9:   end if
10:   $(S_2)_Q \leftarrow (S_2)_Q \cup \{q\}$ 
11:  return  $S_1, S_2, \tilde{p}$ 
12: end if
    
```

Applying the last algorithm to a polynomial p and its partial derivative by its leader yields an algorithm to make polynomials square-free.

In the above ResSplit-based algorithms, we had the requirement that $(S_Q)_{<x}^{\equiv} = \emptyset$. This ensures that all information contained in any equation of a smaller leader than x will be respected by reduction modulo S_T and thus avoids creating redundant systems. It will also be necessary for termination of the Decompose algorithm. This motivates the definition of a selection strategy as follows.

Definition 2.16 (Select). Let $\mathbb{P}_{\text{finite}}(M)$ be the set of all finite subsets of a set M . A **selection strategy** is a map

$$\begin{aligned} \text{Select} : \mathbb{P}_{\text{finite}}(\{p_{=}, p_{\neq} \mid p \in R\}) &\longrightarrow \{p_{=}, p_{\neq} \mid p \in R\} : \\ Q &\longmapsto q \in Q \end{aligned}$$

with the following properties:

- 1. If $\text{Select}(Q) = q_{=}$ is an equation, then $Q_{<\text{ld}(q)}^{\equiv} = \emptyset$.
- 2. If $\text{Select}(Q) = q_{\neq}$ is an inequation, then $Q_{\leq \text{ld}(q)}^{\equiv} = \emptyset$.

The second property of Select could be weakened further, i.e., $Q_{<\text{ld}(q)}^{\equiv} = \emptyset$. However, this would result in redundant calculations in the Decompose algorithm, thus we want all equations of the same leader to be treated first.

The following algorithm is trivial. However, it will be replaced with a more complicated algorithm in §3 when the differential THOMAS decomposition is treated.

Algorithm 2.17 (InsertEquation). *Input:* A system S and an equation $r_{=}$ with $\text{ld}(r) = x$ satisfying $\phi_{\mathbf{a}}(\text{init}(r)) \neq 0$ and $\phi_{<x, \mathbf{a}}(r)$ square-free for all $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$.

Output: A system S where $r_{=}$ is inserted into S_T .

Algorithm:

```

1: if  $(S_T)_x$  is not empty then
2:    $S_T \leftarrow (S_T \setminus \{(S_T)_x\})$ 
3: end if
    
```

```

4:  $S_T \leftarrow S_T \cup \{r_{=}\}$ 
5: return  $S$ 

```

Now we present the main algorithm. It is based on all above algorithms and yields an algebraic THOMAS decomposition. This algorithm forms the basis of the differential THOMAS decomposition to be discussed in detail in §3.

The general structure is as follows: In each iteration, a system S is selected from a list P of unfinished systems. An equation or inequation q is chosen from S_Q according to the selection strategy and reduced modulo S_T . The algorithm assimilates q into S_T , potentially adding inequations of lower leader to S_Q and adding new systems S_i to P that contain a new equation of lower leader in $(S_i)_Q$. This process works differently depending on whether q and $(S_T)_{\text{ld}(q)}$ are equations or inequations, but it is based on the `InitSplit`, `ResSplitGcd` and `ResSplitDivide` methods in all cases. As soon as the algorithm yields an equation $c = 0$ for $c \in F \setminus \{0\}$ or an inequation $0 \neq 0$ in a system, this system is inconsistent and thus discarded.

Algorithm 2.18 (Decompose). The algorithm is printed on page 9.

In the next section, we consider an extension of this algorithm to partial differential systems. Both algorithms have been implemented, and their implementation aspects are considered in §4.

3 The Differential Thomas Decomposition

The differential THOMAS decomposition is concerned with manipulations of polynomial differential equations. The basic idea for a construction of this decomposition is twofold. On the one hand a combinatorial calculus developed by JANET takes care of finding unique reducers and all differential consequences by completing systems to involution. On the other hand the algebraic THOMAS decomposition makes the necessary splits for regularity of initials during the computation and ensures disjointness.

We start by giving the basic definitions from differential algebra needed for the algorithms. Afterwards we summarize the JANET division and its combinatorics. The combinatorics give us a new algorithm `InsertEquation` to add equations into systems. Afterwards we review the differential implications of the algebraic decomposition algorithm and present the algorithm `Reduce` utilized for differential reduction. Replacing the insertion and reduction methods from the algebraic case with these differential methods yields the differential decomposition algorithm.

3.1 Preliminaries from Differential Algebra

Let $\Delta = \{\partial_1, \dots, \partial_n\}$ be the set of derivations ($n > 0$) and F be a computable Δ -**differential field** of characteristic zero. This means any $\partial_j \in \Delta$ is a linear operator $\partial_j : F \rightarrow F$ fulfilling the LEIBNIZ rule. For a **differential indeterminate** u consider the Δ -**differential polynomial ring** $F\{u\} := F[u_i \mid i \in \mathbb{Z}_{\geq 0}^n]$, a polynomial ring infinitely generated by the algebraically independent set $\langle u \rangle_\Delta := \{u_i \mid i \in \mathbb{Z}_{\geq 0}^n\}$. The operation of $\partial_j \in \Delta$ on $\langle u \rangle_\Delta$ by $\partial_j u_i = u_{i+e_j}$ is extended linearly and by the LEIBNIZ rule to $F\{u\}$. Let $U = \{u^{(1)}, \dots, u^{(m)}\}$ be the set of differential indeterminates. The multivariate Δ -differential polynomial ring is given by $F\{U\} := F\{u^{(1)}\} \dots \{u^{(m)}\}$. The elements of $\langle U \rangle_\Delta := \{u_i^{(j)} \mid i \in \mathbb{Z}_{\geq 0}^n, j \in \{1, \dots, m\}\}$ are called **differential variables**.

We remark, that the algebraic closure \overline{F} of F is a differential field with a differential structure uniquely defined by the differential structure of F (cf. [Kol73, §II.2, Lemma 1]). Let

$$E := \bigoplus_{j=1}^m \overline{F}[[z_1, \dots, z_n]] \cong \overline{F}^{(U)\Delta}$$

Algorithm 2.18 (Decompose)

Input: A system S' with $(S')_T = \emptyset$.

Output: A THOMAS decomposition of S' .

Algorithm:

```

1:  $P \leftarrow \{S'\}; \text{Result} \leftarrow \emptyset$ 
2: while  $|P| > 0$  do
3:   Choose  $S \in P$ ;  $P \leftarrow P \setminus \{S\}$ 
4:   if  $|S_Q| = 0$  then
5:      $\text{Result} \leftarrow \text{Result} \cup \{S\}$ 
6:   else
7:      $q \leftarrow \text{Select}(S_Q); S_Q \leftarrow S_Q \setminus \{q\}$ 
8:      $q \leftarrow \text{Reduce}(q, S_T); x \leftarrow \text{ld}(q)$ 
9:     if  $q \notin \{0_{\neq}, c_{=} \mid c \in F \setminus \{0\}\}$  then
10:      if  $x \neq 1$  then
11:        if  $q$  is an equation then
12:          if  $(S_T)_x$  is an equation then
13:            if  $\text{Reduce}(\text{res}_0((S_T)_x, q, x), S_T) = 0$  then
14:               $(S, S_1, p) \leftarrow \text{ResSplitGCD}(S, q, x); P \leftarrow P \cup \{S_1\}$ 
15:               $S \leftarrow \text{InsertEquation}(S, p_{=})$ 
16:            else
17:               $S_Q \leftarrow S_Q \cup \{q_{=}, \text{res}_0((S_T)_x, q, x)_{=}\}$ 
18:            end if
19:          else
20:            if  $(S_T)_x$  is an inequationa then
21:               $S_Q \leftarrow S_Q \cup \{(S_T)_x\}; S_T \leftarrow S_T \setminus \{(S_T)_x\}$ 
22:            end if
23:             $(S, S_2) \leftarrow \text{InitSplit}(S, q); P \leftarrow P \cup \{S_2\}$ 
24:             $(S, S_3, p) \leftarrow \text{ResSplitDivide}(S, q, \frac{\partial}{\partial x} q); P \leftarrow P \cup \{S_3\}$ 
25:             $S \leftarrow \text{InsertEquation}(S, p_{=})$ 
26:          end if
27:          else if  $q$  is an inequation then
28:            if  $(S_T)_x$  is an equation then
29:               $(S, S_4, p) \leftarrow \text{ResSplitDivide}(S, (S_T)_x, q); P \leftarrow P \cup \{S_4\}$ 
30:               $S \leftarrow \text{InsertEquation}(S, p_{=})$ 
31:            else
32:               $(S, S_5) \leftarrow \text{InitSplit}(S, q); P \leftarrow P \cup \{S_5\}$ 
33:               $(S, S_6, p) \leftarrow \text{ResSplitDivide}(S, q, \frac{\partial}{\partial x} q); P \leftarrow P \cup \{S_6\}$ 
34:              if  $(S_T)_x$  is an inequation then
35:                 $(S, S_7, r) \leftarrow \text{ResSplitDivide}(S, (S_T)_x, p); P \leftarrow P \cup \{S_7\}$ 
36:                 $(S_T)_x \leftarrow (r \cdot p)_{\neq}$ 
37:              else if  $(S_T)_x$  is empty then
38:                 $(S_T)_x \leftarrow p_{\neq}$ 
39:              end if
40:            end if
41:          end if
42:        end if
43:         $P \leftarrow P \cup \{S\}$ 
44:      end if
45:    end if
46:  end while
47: return  $\text{Result}$ 

```

^a Remember that $(S_T)_x$ might be empty, and thus neither an equation nor an inequation.

with indeterminates z_1, \dots, z_n , where $\overline{F}[[z_1, \dots, z_n]]$ denotes the ring of formal power series. The isomorphism maps coefficients of the power series to function values of differential variables, i.e.,

$$\alpha : \bigoplus_{i=1}^m \overline{F}[[z_1, \dots, z_n]] \rightarrow \overline{F}^{(U)\Delta} : \left(\sum_{\mathbf{i} \in \mathbb{Z}_{\geq 0}^n} a_{\mathbf{i}}^{(1)} \frac{z^{\mathbf{i}}}{\mathbf{i}!}, \dots, \sum_{\mathbf{i} \in \mathbb{Z}_{\geq 0}^n} a_{\mathbf{i}}^{(m)} \frac{z^{\mathbf{i}}}{\mathbf{i}!} \right) \mapsto \left(u_{\mathbf{i}}^{(j)} \mapsto a_{\mathbf{i}}^{(j)} \right)$$

where $\mathbf{i}! := i_1! \cdot \dots \cdot i_n!$ defines the factorial of a multi-index.

In the formulation of the algorithm the direct sum of formal power series E suffices to give a notion of solutions coherent to the algebraic case: For $e \in E$ we define the F -algebra homomorphism

$$\phi_e : F\{U\} \rightarrow \overline{F} : u_{\mathbf{i}}^{(j)} \mapsto \alpha(e)(u^{(j)})$$

evaluating all differential variables of a differential polynomial at the power series e . A **differential equation** or **inequation** for m functions $U = \{u^{(1)}, \dots, u^{(m)}\}$ in n indeterminates is an element $p \in F\{U\}$ written $p =$ or $p \neq$, respectively. A **solution** of $p =$ or $p \neq$ is an $e \in E$ with $\phi_e(p) = 0$ or $\phi_e(p) \neq 0$, respectively. More generally $e \in E$ is called a solution of a set P of equations and inequations, if it is a solution of each element in P . The set of solutions of P is denoted by $\mathfrak{Sol}_E(P) = \mathfrak{Sol}(P) \subseteq E$. Since we substitute elements of \overline{F} algebraically for the differential indeterminates, Remark (2.2), which guarantees the continuation of solutions from lower ranking variables to higher ranking ones, also holds here.

Any differential F -algebra R with a differential embedding of $E \hookrightarrow R$ might be chosen as universal set of solutions, for example a universal differential field containing F : Clearly $\overline{F}[[z_1, \dots, z_n]]$ embeds into its field of quotients $\overline{F}((z_1, \dots, z_n))$, and thus $\overline{F}[[z_1, \dots, z_n]]$ also embeds into a universal differential field containing F , since $\overline{F}((z_1, \dots, z_n))$ is a finitely generated differential field extension of \overline{F} (cf. [Kol73, §II.2 and §III.7]). We denote the set of solutions in R by $\mathfrak{Sol}_R(P) \subseteq R$.

A finite set of equations and inequations is called a **(differential) system** over $F\{U\}$. We will be using the same notation for systems as in the algebraic THOMAS decomposition introduced in §2.1 and §2.2, in particular a system S is represented by a pair (S_T, S_Q) . However, the candidate simple system S_T will also reflect a differential structure using combinatorial methods. We will elaborate on the combinatorics in the next section.

3.2 The Combinatorics of Janet Division

In this subsection we will focus on the combinatorics of equations, enabling us to control the infinite set of differential variables appearing as partial derivatives of differential indeterminates. For this purpose we use JANET division [GB98a] which defines these combinatorics and also automatizes construction of integrability conditions. An overview of modern development can be found in [Ger05, Sei10] and the original ideas by JANET are formulated in [Jan29]. This is achieved by partitioning the set of differential variables into finitely many “cones” and “free” variables. For creating this partition we present an algorithm for inserting new equations into an existing set of equations and adjusting the cone decomposition. Apart from this insertion algorithm the only other adaptation of the algebraic **Decompose** algorithm (2.18) will be the reduction algorithm presented in §3.3.

We fix a (differential) **ranking** $<$, which is defined as a total order on the differential variables such that $u^{(k)} < \partial_j u^{(k)}$ and $u^{(k)} < u^{(l)}$ implies $\partial_j u^{(k)} < \partial_j u^{(l)}$ for all $u^{(k)}, u^{(l)} \in U$, $\partial_j \in \Delta$. For any finite set of differential variables, a differential ranking is a ranking as defined for the algebraic case in §2.1. This allows us to define the largest differential variable $\text{ld}(p)$ appearing in a differential polynomial $p \in F\{U\}$ as **leader**, which is set to 1 for $p \in F$. Furthermore, define $\text{rank}(p)$ and $\text{init}(p)$ as the degree in the leader and the coefficient of $\text{ld}(p)^{\text{rank}(p)}$, respectively. Again we will assume $1 < u_{\mathbf{i}}^{(j)}$ for all $j \in \{1, \dots, m\}$ and $\mathbf{i} \in \mathbb{Z}_{\geq 0}^n$.

A set W of differential variables is **closed** under the action of $\Delta' \subseteq \Delta$ if $\partial_i w \in W \forall \partial_i \in \Delta', w \in W$. The smallest such closed set containing a differential variable w denoted by $\langle w \rangle_{\Delta'}$ is called a

cone and the elements of Δ' we shall call (JANET) **admissible derivations**⁴. The Δ' -closed set generated by a set W of differential variables is defined to be

$$\langle W \rangle_{\Delta'} := \bigcap_{\substack{W_i \supseteq W \\ W_i \Delta'\text{-closed}}} W_i \subseteq \langle U \rangle_{\Delta}.$$

For a finite set $W = \{w_1, \dots, w_r\}$, the JANET **division** algorithmically assigns admissible derivations to the elements of W such that the cones generated by the $w \in W$ are disjoint. The derivation $\partial_l \in \Delta$ is assigned to the cone generated by $w = u_1^{(j)} \in W$ as admissible derivation, if and only if

$$\mathbf{i}_l = \max \left\{ \mathbf{i}'_l \mid u_{\mathbf{i}'}^{(j)} \in W, \mathbf{i}'_k = \mathbf{i}_k \text{ for all } 1 \leq k < l \right\}$$

holds. We remark, that j is fixed in this definition, i.e., when constructing cones we only take into account other differential variables belonging to the same differential indeterminate. The admissible derivations assigned to w are denoted by $\Delta_W(w) \subseteq \Delta$ and we call the cone $\langle w \rangle_{\Delta_W(w)}$ the JANET **cone** of w with respect to W . This construction ensures disjointness of cones but not necessarily that the union of cones equals $\langle W \rangle_{\Delta}$. For the JANET **completion** a finite set $\widetilde{W} \supset W$ is successively created by adding any $\tilde{w} = \partial_i w_j \notin \biguplus_{w \in \widetilde{W}} \langle w \rangle_{\Delta_{\widetilde{W}}(w)}$ to \widetilde{W} , where $w_j \in \widetilde{W}$ and $\partial_i \in \Delta \setminus \Delta_{\widetilde{W}}(w_j)$. This leads to the disjoint JANET **decomposition**

$$\langle W \rangle_{\Delta} = \biguplus_{w \in \widetilde{W}} \langle w \rangle_{\Delta_{\widetilde{W}}(w)}$$

that separates a Δ -closed set W into finitely many cones $\langle w \rangle_{\Delta_{\widetilde{W}}(w)}$ after finitely many steps. For details see [Ger05, Def. 3.4] and [GB98a, Corr. 4.11].

With the JANET decomposition being defined for sets of differential variables, we will assign admissible derivations to differential polynomials according to their leaders. In particular, we extend the definitions of $\Delta_W(w)$ for finite $W \subset F\{U\}$ and $w \in W$ by defining $\Delta_W(w) := \Delta_{\text{ld}(W)}(\text{ld}(w))$.

A differential polynomial $q \in F\{U\}$ is called **reducible** with respect to $p \in F\{U\}$, if there exists $\mathbf{i} \in \mathbb{Z}_{\geq 0}^n$ such that $\partial_1^{\mathbf{i}_1} \dots \partial_n^{\mathbf{i}_n} \text{ld}(p) = \text{ld}(\partial_1^{\mathbf{i}_1} \dots \partial_n^{\mathbf{i}_n} p) = \text{ld}(q)$ and $\text{rank}(\partial_1^{\mathbf{i}_1} \dots \partial_n^{\mathbf{i}_n} p) \leq \text{rank}(q)$. We call a derivative of an equation by an admissible derivation an **admissible prolongation**. When restricting ourselves to admissible prolongation, we get the following concept: For a finite set $T \subset F\{U\}$ we call a differential polynomial $q \in F\{U\}$ JANET **reducible** with respect to $p \in T$, if there exists $\mathbf{i} \in \mathbb{Z}_{\geq 0}^n$ such that $\partial_1^{\mathbf{i}_1} \dots \partial_n^{\mathbf{i}_n} \text{ld}(p) = \text{ld}(q)$ with all proper derivatives being admissible and $\text{rank}(\partial_1^{\mathbf{i}_1} \dots \partial_n^{\mathbf{i}_n} p) \leq \text{rank}(q)$ holds. We shall also say that q is JANET **reducible** modulo T if there is a $p \in T$, such that q is JANET reducible with respect to $p \in T$.

A set of differential variables $T \subset \langle U \rangle_{\Delta}$ is called **minimal**, if for any set $S \subset \langle U \rangle_{\Delta}$ with $\biguplus_{t \in T} \langle t \rangle_{\Delta_T(t)} = \biguplus_{s \in S} \langle s \rangle_{\Delta_S(s)}$ the condition $T \subseteq S$ holds (cf. [GB98b, Def. 4.2]). We also call a set of differential polynomials minimal, if the corresponding set of leaders is minimal.

In addition to the non-zero initials and square-freeness of polynomials in the candidate set S_T for a simple system (as defined in §2.2), the equations in $(S_T)^=$ are required to have admissible derivations assigned to them. When an equation p is not reducible modulo $(S_T)^=$ it is added to $(S_T)^=$ and all polynomials in S_T with a leader being a derivative of $\text{ld}(p)$ are removed from S_T , ensuring minimality. Furthermore, all non-admissible prolongations are created to be processed. This is formulated in the following algorithm:

Algorithm 3.1 (InsertEquation).

Input: A system S' and a polynomial $p_{=} \in F\{U\}$ not reducible modulo $(S_T')^=$.
Output: A system S , where $(S_T)^= \subseteq (S_T')^= \cup \{p_{=}\}$ is maximal satisfying

$$\{\text{ld}(q) \mid q \in (S_T) \setminus \{p_{=}\}\} \cap \langle \text{ld}(p_{=}) \rangle_{\Delta} = \emptyset,$$

$$S_Q = S'_Q \cup (S'_T \setminus S_T) \cup \{(\partial_i q)_{=} \mid q \in (S_T)^=, \partial_i \notin \Delta_{((S_T)^=)}(q)\}.$$

Algorithm:

⁴ In [Ger99] and [Sei10, Chap. 7] the admissible derivations are called (JANET) multiplicative variables.

```

1:  $S \leftarrow S'$ 
2:  $S_T \leftarrow S_T \cup \{p=\}$ 
3: for  $q \in S_T \setminus \{p\}$  do
4:   if  $\text{ld}(q) \in \langle \text{ld}(p) \rangle_\Delta$  then
5:      $S_Q \leftarrow S_Q \cup \{q\}$ 
6:      $S_T \leftarrow S_T \setminus \{q\}$ 
7:   end if
8: end for
9: Reassign admissible derivations to  $(S_T)^\perp$ 
10:  $S_Q \leftarrow S_Q \cup \{(\partial_i q)_= \mid q \in (S_T)^\perp, \partial_i \notin \Delta_{((S_T)^\perp)}(q)\}$ 
11: return  $S$ 

```

We remark that a non-admissible prolongations might be added to S_Q again each step, even though it has been added before. This can be prevented by simply storing all previously generated non-admissible prolongations.

3.3 Differential Simple Systems

This section goes on reducing the differential decomposition algorithm to the algebraic one. We start by introducing partial solutions in order to algebraically evaluate differential polynomials at them yielding univariate differential polynomials. Then we present a differential reduction algorithm, as the second distinction from the algebraic decomposition algorithm. At last this section defines differential simple systems.

For a differential variable $x \in \langle U \rangle_\Delta$ and a power series $e \in E$ define the F -algebra homomorphism

$$\phi_{<x,e} : F\{U\} \rightarrow \overline{F}[v \mid v \in \langle U \rangle_\Delta, v \geq x] : \begin{cases} u_{\mathbf{i}}^{(j)} \mapsto \alpha(e)(u_{\mathbf{i}}^{(j)}), & \text{for } u_{\mathbf{i}}^{(j)} < x \\ u_{\mathbf{i}}^{(j)} \mapsto u_{\mathbf{i}}^{(j)}, & \text{for } u_{\mathbf{i}}^{(j)} \geq x \end{cases}$$

evaluating all differential variables of a differential polynomial at e which are $<$ -smaller than x .

For differential reduction the JANET partition of differential variables provides the mechanism to get a unique reductor in a fast way (for an algorithm see [GYB01]) by restricting to admissible prolongations. After finding a reductor we apply a pseudo remainder algorithm (see Eq. (1)).

We need to ensure that initials (and initials of the derivatives) of equations are non-zero. Let $p \in F\{U\}$ with $x = \text{ld}(p)$ and define the **separant** $\text{sep}(p) := \frac{\partial p}{\partial x}$. One easily checks (cf. [Kol73, §I.8, lemma 5] or [Hub03b, §3.1]) that the initial of any derivative of p is $\text{sep}(p)$ and the separant of any square-free equation p is nonzero on $\mathfrak{Sol}(p)$. So by making the equations square-free, it is ensured that pseudo reductions are not only possible modulo p , but also modulo its derivatives. This allows us to formulate the differential reduction algorithm:

Algorithm 3.2 (Reduce).

Input: A differential system S and a polynomial $p \in F\{U\}$.

Output: A polynomial q that is not JANET reducible modulo S_T with $\phi_e(p) = 0$ if and only if $\phi_e(q) = 0$ for each $e \in \mathfrak{Sol}(S)$.

Algorithm:

```

1:  $x \leftarrow \text{ld}(p)$ 
2: while exists  $q_= \in (S_T)^\perp$  and  $i_1, \dots, i_n \in \mathbb{Z}_{\geq 0}$  with  $i_j = 0$  for  $\partial_j \notin \Delta_{(S_T)^\perp}(q)$  such that
    $\partial_1^{i_1} \dots \partial_n^{i_n} \text{ld}(q) = \text{ld}(p)$  and  $\text{rank}(\partial_1^{i_1} \dots \partial_n^{i_n} p) \geq \text{rank}(q)$  do
3:    $p \leftarrow \text{prem}(p, \partial_1^{i_1} \dots \partial_n^{i_n} q, x)$ 
4:    $x \leftarrow \text{ld}(p)$ 
5: end while
6: if  $\text{Reduce}(S, \text{init}(p)) = 0$  then
7:   return  $\text{Reduce}(S, p - \text{init}(p)x^{\text{rank}(p)})$ 
8: else

```

9: **return** p
 10: **end if**

A polynomial $p \in F\{U\}$ is called **reduced**⁵ **modulo** S_T if $\text{Reduce}(S, p) = p$. A polynomial $p \in F\{U\}$ **reduces to** q **modulo** S_T if $\text{Reduce}(S, p) = q$.

Usually in differential algebra, one distinguishes a (full) differential reduction as used here and a partial (differential) reduction. Partial reduction only employs *proper* derivations of equations for reduction (cf. [Kol73, §I.9] or [Hub03b, §3.2]). This is useful for separation of differential and algebraic parts of the algorithm and for the use of ROSENFELD's Lemma (cf. [Ros59]).

Definition 3.3 (Differential Simple Systems). A differential system S is (JANET) **involutive**, if all non-admissible prolongations in $(S_T)^\perp$ reduce to zero by $(S_T)^\perp$.

A system S is called **differentially simple** or **simple**, if S is

- a) algebraically simple in the finite set of differential variables appearing in it,
- b) involutive,
- c) S^\perp is minimal,
- d) no inequation is reducible modulo S^\perp .

A disjoint decomposition of a system into differentially simple subsystems is called **(differential) THOMAS decomposition**.

3.4 The Differential Decomposition Algorithm

The differential THOMAS decomposition algorithm is a modification of the algebraic THOMAS decomposition algorithm. We have already introduced the new algorithms **InsertEquation** (3.1) for adding new equations into the systems and **Reduce** (3.2) for reduction, that can replace their counterparts in the algebraic algorithm.

Algorithm 3.4 (DifferentialDecompose).

Input: A differential system S' with $(S')_T = \emptyset$.

Output: A differential THOMAS decomposition of S' .

Algorithm: The algorithm is obtained by replacing the two subalgorithms **InsertEquation** and **Reduce** in (2.18) with their differential counterparts (3.1) and (3.2), respectively.

We give an example taken from [BC99, pp. 597-600]:

Example 3.5 (Cole-Hopf Transformation). For $F := \mathbb{R}(x, t)$, $\Delta = \{\frac{\partial}{\partial x}, \frac{\partial}{\partial t}\}$, and $U = \{\eta, \zeta\}$ consider the heat equation $h = \eta_t + \eta_{xx} \in F\{U\}^\perp$ and BURGER's equation $b = \zeta_t + \zeta_{xx} + 2\zeta_x \cdot \zeta \in F\{U\}^\perp$. To improve readability, leaders of polynomials are underlined below.

First we claim that any power series solution for the heat equation with a non-zero constant term can be transformed to a solution of BURGER's equation by means of the COLE-HOPF transformation $\lambda : \eta \mapsto \frac{\eta_x}{\eta}$. The differential THOMAS decomposition for an orderly ranking with $\zeta_x > \eta_t$ of

$$\{h, \underbrace{(\eta \cdot \zeta - \eta_x)}_{\Leftrightarrow \zeta = \lambda(\eta)}, \eta_\neq\}$$

consists of the single system

$$S = \{(\underline{\eta_x} - \eta \cdot \zeta), (\eta \cdot \underline{\zeta_x} + \eta_t + \eta \cdot \zeta^2), \underline{\eta_\neq}\}$$

and one checks that $\text{Reduce}(S, b) = 0$ holds. This implies that any non-zero solution of the heat equation is mapped by the COLE-HOPF transformation to a solution of BURGER's equation.

⁵ There is a fine difference between not being reducible and being reduced. In the case of not being reducible the initial of a polynomial can still reduce to zero and iteratively the entire polynomial.

In addition we claim that λ is surjective. For the proof we choose an elimination ranking (cf. [Hub03b, §8.1] or [Bou07]) with $\eta \gg \zeta$, i.e., $\eta_{\mathbf{i}} > \zeta_{\mathbf{j}}$ for all $\mathbf{i}, \mathbf{j} \in (\mathbb{Z}_{\geq 0})^2$. We compute the differential THOMAS decomposition of $\{h_-, b_-, (\eta \cdot \zeta - \eta_x)_-, \eta_{\neq}\}$ which again consists of a single system

$$S = \{(\underline{\eta}_x - \eta \cdot \zeta)_-, (\eta \cdot \zeta_x + \underline{\eta}_t + \eta \cdot \zeta^2)_-, \underline{b}_-, \underline{\zeta}_{\neq}\}.$$

The properties of a simple system ensure that for any solution of lower ranking equations there exists a solution of the other equations (cf. (2.2)). The elimination ordering guarantees that the only constraint for ζ is BURGER's equation b_- and thus for any solution $f \in \mathfrak{Sol}(b_-)$ there exists a solution $(g, f) \in \mathfrak{Sol}(S)$. Furthermore, since h_- was added to the input system, $g \in \mathfrak{Sol}(h_-)$ holds and finally the equation $(\eta \cdot \zeta - \eta_x)_-$ implies $\lambda(g) = f$.

Remark 3.6. Elements of the differential field are not subjected to splittings, unless they are modelled as differential indeterminates. For example to model a differential field $F = \mathbb{C}(x)$ with $\Delta = \{\frac{\partial}{\partial x}\}$, we add an extra differential indeterminate X to U and replace x by X in all equations and inequations. We subject X to the relation $\frac{\partial}{\partial x}X = 1$ for X being “generic” or $(\frac{\partial}{\partial x}X - 1) \cdot \frac{\partial}{\partial x}X = 0$, if we allow X to degenerate to a point. This will be subject of further study.

4 Implementation

4.1 Algorithmic Optimizations

In the `Decompose` algorithm, pseudo remainder sequences for the same pairs of polynomials are usually needed several times. As these calculations are expensive in general, for *avoiding repeated calculations*, it is important that the results are kept in memory and will be reused when the same sequence is requested again.

If a polynomial admits *factorization*, we can use it to save computation time. More precisely, a disjoint decomposition of the system $S \uplus \{(p \cdot q)_-\}$ is given by $(S \cup \{p_-\}, S \cup \{p_{\neq}, q_-\})$ and the system $S \uplus \{(p \cdot q)_{\neq}\}$ is equivalent to $S \cup \{p_{\neq}, q_{\neq}\}$. Let $Y_i := \{x_j \mid x_j < x_i, (S_T)_{x_j}^- \neq \emptyset\}$ and $Z_i := \{x_j \mid x_j < x_i, (S_T)_{x_j}^- = \emptyset\}$. If $(S_T)_{x_i}^-$ is irreducible over the field $F_i := F(Z_i)[Y_i]/\langle (S_T)_{x_i}^- \rangle_{F(Z_i)[Y_i]}$ for all $i \in \{1, \dots, n\}$, where $\langle (S_T)_{x_i}^- \rangle_{F(Z_i)[Y_i]}$ is the ideal generated by $(S_T)_{x_i}^-$ in the polynomial ring $F(Z_i)[Y_i]$, factorization of polynomials can be performed over F_n instead of F .

Coefficient growth is a common problem in elimination. If possible, polynomials should be represented as compact as possible. Once it is known that the initial cannot vanish, the *content* (in the univariate sense) cannot vanish either. Thus, every time an initial has been added as an inequation to the system, one can divide the polynomial by its content.

If the ground field F is represented as a field of fractions of a domain D (like the rationals or a rational function field over the rationals), it also makes sense to remove the multivariate content, which is an element of F .

When reducing, in addition to reduction modulo the polynomial of the same leader, reducing the coefficients modulo the polynomials of lower leader can be considered. In some cases this leads to a reduction of sizes of coefficients, in other cases sizes increase. The latter is partly due to whole polynomials being multiplied with initials of the reducers. Finding a good heuristic for this *coefficient reduction* is crucial for efficiency.

In the algebraic algorithm, polynomials don't necessarily have to be square-free when they are inserted into the candidate simple system. Efficiency is sometimes improved greatly by postponing the calculation of the square-free split as long as possible.

In the differential case application of *criteria* to avoid useless reduction of non-admissible prolongations can decrease computation time. The combinatorial approach used in this paper already avoids many reductions of so-called Δ -polynomials, as used in other approaches (see [GY06]). Nonetheless, using the involutive criteria 2-4 (cf. [GB98a, Ger05, AH05] and [BLOP09, §4, Prop. 5]) which together are equivalent to the chain criterion, is valid and helpful.

Another possible improvement is parallelization, since the main loop in line 2 of `Decompose` (2.18) can naturally be used in parallel for different systems.

4.2 Implementation in MAPLE

Both algorithms have been implemented in the MAPLE computer algebra system. Packages can be downloaded from [BLH10], documentation and example worksheets are available there.

The main reason for choosing MAPLE for the implementation is the collection of solvers for polynomial equations, ODEs, and PDEs already present. Furthermore, fast algorithms exist for polynomial factorization over finitely generated field extensions of \mathbb{Q} and for gcd computation. Computation of subresultants is not available in MAPLE, therefore an algorithm based on [Duc00] is implemented for that purpose.

Features for the differential package include arbitrary differential rankings, using functions implemented in MAPLE as differential field, computation of power series solutions, and a direct connection to the solvers of MAPLE for differential equations.

Example 4.1. Start by loading the current version of our package:

```
> with(DifferentialThomas):
> ComputeRanking([t],[x2,x1,y,u],"EliminateFunction");
```

This creates the differential polynomial ring $\mathbb{Q}\{x^{(2)}, x^{(1)}, y, u\}$ for $\Delta = \{\frac{\partial}{\partial t}\}$. Here u indicates the input, $x^{(1)}$ and $x^{(2)}$ the state, and y the output of the system. The chosen ranking “<” is the elimination ranking with $x^{(2)} \gg x^{(1)} \gg y \gg u$, i.e., $x_i^{(2)} > x_j^{(1)} > y_k > u_l$ for all $i, j, k, l \in \mathbb{Z}_{\geq 0}$.

```
> L:=[x1[1]-u[0]*x2[0],x2[1]-x1[0]-u[0]*x2[0],y[0]-x1[0]];
      L := [x1_1 - u_0 x2_0, x2_1 - x1_0 - u_0 x2_0, y_0 - x1_0]
```

We follow [Dio92, Ex. 1] and want to compute the external trajectories of a differential ideal generated by L , i.e. intersect this differential ideal with $\mathbb{Q}\{y, u\}$.

```
> res:=DifferentialThomasDecomposition(L,[]);
      res := [DifferentialSystem, DifferentialSystem]
```

We show the equations and inequations of the differential systems not involving $x^{(1)}$ and not involving $x^{(2)}$. The chosen ranking guarantees that for each differential system of the output, all constraints holding for lower ranking differential indeterminates can be read off the equations and inequations only involving these differential indeterminates, i.e., the systems shown determine the external trajectories of the system:

```
> PrettyPrintDifferentialSystem(res[1]):
> remove(a->has(a,x2) or has(a,x1),%);
      [-u(t)(\frac{d^2}{dt^2} y(t)) + (\frac{d}{dt} y(t)) u(t)^2 + (\frac{d}{dt} y(t)) (\frac{d}{dt} u(t)) + y(t) u(t)^2 = 0, u(t) \neq 0]
> PrettyPrintDifferentialSystem(res[2]):
> remove(a->has(a,x2) or has(a,x1),%);
      [\frac{d}{dt} y(t) = 0, u(t) = 0]
```

These systems, having disjoint solution sets, are identical to the ones found in [Dio92].

4.3 Implementations of Similar Decomposition Algorithms

The **RegularChains** package [LMX05], which is shipped with recent versions of MAPLE, implements a decomposition of a polynomial ideal into ideals represented by regular chains and a radical decomposition of an ideal into square-free regular chains. The solution sets of this decomposition are in general not disjoint. However, there is an extension called comprehensive triangular decomposition (cf. [CGL⁺07]) that provides disjointness on the parameters of a parametric system. The systems of the parameters are not simple systems though. The **RegularChains** package contains **FastArithmeticTools** as a subpackage implementing asymptotically fast polynomial arithmetic for the modular case.

The `epsilon` package ([Wan03]) by Dongming Wang implements different kinds of triangular decompositions (including a decomposition into regular chains like above) in MAPLE. It is the only software package besides our own that implements the THOMAS decomposition. It uses the simpler “top-down” approach that Thomas (cf. [Tho37, Tho62]) suggested, i.e., polynomials of higher leader are considered first. All polynomials of the same leader are combined into one common consequence. New systems, which contain conditions on initials of polynomials and subresultants, are created by splitting subalgorithms similar to ours. All these new conditions of lower leader are not taken into account for now and will be treated in a later step. Contrary to our approach, one cannot reduce modulo an *unfinished* system and hence inconsistency checks are less natural and more complicated. It is conceivable that this strategy spends too much time on computations with inconsistent systems. Therefore, `epsilon` implements highly sophisticated heuristics for early detection of inconsistent systems. It achieves similar performance to our implementation.

The MAPLE package `difalg` [BH04] deals with ordinary and partial differential equations as described in [BLOP09]. Its functionalities are used by symbolic differential equations solvers in MAPLE. For an input of equations and inequations it computes a radical decomposition of the differential ideal generated by the equations and saturated by the inequations. I.e., a description of the vanishing ideal of the KOLCHIN closure (cf. [Kol73, §IV.1]) of the solutions is computed. The output are differential characteristic sets as introduced by RITT [Rit50, §1.5]. Computation of differential consequences is driven by reduction of Δ -polynomials, which are the analogon of s -polynomials in differential algebra. We found the system being optimized and well-suited for computations with ordinary differential equations.

Similar algorithms as in `difalg` are used in the BLAD-libraries [Bou09]. It is designed as a stand-alone C-library with an emphasis on usability for non-mathematicians and extensive documentation. As it is written in C, BLAD is expected to outperform `difalg` for relevant examples.

For future publications, we plan to compare the THOMAS decomposition and our implementation with other decompositions and implementations. We also plan to further examine applications that benefit from the properties of simple systems.

5 Acknowledgements

The contents of this paper profited very much from numerous useful comments and remarks from Wilhelm Plesken. The authors thank him as well as Dongming Wang and François Boulier for fruitful discussions. Furthermore, our gratitude goes to the anonymous referees for valuable comments and for pointing out informative references. The second author (V.P.G.) acknowledges the Deutsche Forschungsgemeinschaft for the financial support that made his stay in Aachen possible. The presented results were obtained during his visits.

References

- [AH05] Joachim Apel and Ralf Hemmecke, *Detecting unnecessary reductions in an involutive basis computation*, J. Symbolic Comput. **40** (2005), no. 4-5, 1131–1149. MR MR2169107 (2006j:13026) 14
- [BC99] Alexandru Buium and Phyllis J. Cassidy, *Differential algebraic geometry and differential algebraic groups: from algebraic differential equations to diophantine geometry*, in [Kol99] (1999), 567–636. 13
- [BCG⁺03] Y. A. Blinkov, C. F. Cid, V. P. Gerdt, W. Plesken, and D. Robertz, *The MAPLE Package JANET: I. Polynomial Systems. II. Linear Partial Differential Equations*, Proc. 6th Int. Workshop on Computer Algebra in Scientific Computing, Passau, Germany, 2003, (<http://wwwb.math.rwth-aachen.de/Janet>), pp. 31–40 and 41–54. 2
- [BH04] François Boulier and Evelyne Hubert, *DIFFALG: description, help pages and examples of use*, 1996-2004, Symbolic Computation Group, University of Waterloo, Ontario, Canada (<http://www-sop.inria.fr/members/Evelyne.Hubert/difalg/>). 1, 16
- [BKRM01] Driss Bouziane, Abdelilah Kandri Rody, and Hamid Maàrouf, *Unmixed-dimensional decomposition of a finitely generated perfect differential ideal*, J. Symbolic Comput. **31** (2001), no. 6, 631–649. MR MR1834002 (2002c:12007) 1

- [BLH10] Thomas Bächler and Markus Lange-Hegermann, *Algebraic Thomas and Differential/Thomas: Thomas decomposition for algebraic and differential systems*, 2008–2010, (<http://wwwb.math.rwth-aachen.de/thomasdecomposition/>). 15
- [BLOP95] François Boulier, Daniel Lazard, François Ollivier, and Michel Petitot, *Representation for the radical of a finitely generated differential ideal*, ISSAC, 1995, pp. 158–166. 1
- [BLOP09] ———, *Computing representations for radicals of finitely generated differential ideals*, Appl. Algebra Engrg. Comm. Comput. **20** (2009), no. 1, 73–121. MR MR2496662 (2010c:12005) 1, 14, 16
- [Bou07] François Boulier, *Differential elimination and biological modelling*, Gröbner bases in symbolic analysis, Radon Ser. Comput. Appl. Math., vol. 2, Walter de Gruyter, Berlin, 2007, pp. 109–137. MR MR2394771 (2009f:12005) 14
- [Bou09] ———, *BLAD: Bibliothèques lilloises d’algèbre différentielle*, 2004–2009, (<http://www.lifl.fr/~boulier/BLAD/>). 1, 16
- [CGL⁺07] Changbo Chen, Oleg Golubitsky, François Lemaire, Marc Moreno Maza, and Wei Pan, *Comprehensive triangular decomposition*, CASC (Victor G. Ganzha, Ernst W. Mayr, and Evgenii V. Vorozhtsov, eds.), Lecture Notes in Computer Science, vol. 4770, Springer, 2007, pp. 73–101. 15
- [Del00] Stéphane Dellière, *D.m. wang simple systems and dynamic constructible closure*, Rapport de Recherche No. 2000–16 de l’Université de Limoges (2000). 1
- [Dio92] Sette Diop, *On universal observability*, Proc. 31st Conference on Decision and Control (Tucson, Arizona), 1992. 15
- [Duc00] Lionel Ducos, *Optimizations of the subresultant algorithm*, J. Pure Appl. Algebra **145** (2000), no. 2, 149–163. MR MR1733249 (2000m:68187) 15
- [GB98a] Vladimir P. Gerdt and Yuri A. Blinkov, *Involutive bases of polynomial ideals*, Math. Comput. Simulation **45** (1998), no. 5–6, 519–541, Simplification of systems of algebraic and differential equations with applications. MR MR1627129 (99e:13033) 1, 2, 10, 11, 14
- [GB98b] ———, *Minimal involutive bases*, Math. Comput. Simulation **45** (1998), no. 5–6, 543–560, Simplification of systems of algebraic and differential equations with applications. MR MR1627130 (99e:13034) 11
- [Ger99] Vladimir P. Gerdt, *Completion of linear differential systems to involution*, Computer algebra in scientific computing—CASC’99 (Munich), Springer, Berlin, 1999, pp. 115–137. MR MR1729618 (2001d:12010) 2, 10
- [Ger05] ———, *Involutive algorithms for computing Gröbner bases*, Computational commutative and non-commutative algebraic geometry, NATO Sci. Ser. III Comput. Syst. Sci., vol. 196, IOS, Amsterdam, 2005, pp. 199–225. MR MR2179201 (2007c:13040) 2, 10, 11, 14
- [Ger08] ———, *On decomposition of algebraic PDE systems into simple subsystems*, Acta Appl. Math. **101** (2008), no. 1–3, 39–51. MR MR2383543 (2009c:35003) 2
- [GY06] Vladimir P. Gerdt and Denis A. Yanovich, *Investigation of the effectiveness of involutive criteria for computing polynomial Janet bases*, Programming and Computer Software **32** (2006), no. 3, 134–138. MR MR2267374 (2007e:13036) 14
- [GYB01] Vladimir P. Gerdt, Denis A. Yanovich, and Yuri A. Blinkov, *Fast search for the Janet divisor*, Programming and Computer Software **27** (2001), no. 1, 22–24. MR MR1867717 12
- [Hab48] Walter Habicht, *Eine Verallgemeinerung des Sturmschen Wurzelzählverfahrens*, Comment. Math. Helv. **21** (1948), 99–116. MR MR0023796 (9,405f) 5
- [Hub03a] Evelyn Hubert, *Notes on triangular sets and triangulation-decomposition algorithms. I. Polynomial systems*, Symbolic and numerical scientific computation (Hagenberg, 2001), Lecture Notes in Comput. Sci., vol. 2630, Springer, Berlin, 2003, pp. 1–39. MR MR2043699 (2005c:13034) 1
- [Hub03b] ———, *Notes on triangular sets and triangulation-decomposition algorithms. II. Differential systems*, Symbolic and numerical scientific computation (Hagenberg, 2001), Lecture Notes in Comput. Sci., vol. 2630, Springer, Berlin, 2003, pp. 40–87. MR MR2043700 (2005c:13035) 1, 12, 13, 14
- [Jan29] Maurice Janet, *Leçons sur les systèmes des équationes aux dérivées partielles*, Cahiers Scientifiques IV, Gauthiers-Villars, Paris, 1929. 1, 10
- [Kol73] Ellis R. Kolchin, *Differential algebra and algebraic groups*, Academic Press, New York, 1973, Pure and Applied Mathematics, Vol. 54. MR MR0568864 (58 #27929) 8, 10, 12, 13, 16
- [Kol99] ———, *Selected works of Ellis Kolchin with commentary*, American Mathematical Society, Providence, RI, 1999, Commentaries by Armand Borel, Michael F. Singer, Bruno Poizat, Alexandru Buium and Phyllis J. Cassidy, Edited and with a preface by Hyman Bass, Buium and Cassidy. MR MR1677530 (2000g:01042) 16

- [LMX05] F. Lemaire, M. Moreno Maza, and Y. Xie, *The RegularChains library in MAPLE*, SIGSAM Bull. **39** (2005), no. 3, 96–97. 15
- [LW99] Ziming Li and Dongming Wang, *Coherent, regular and simple systems in zero decompositions of partial differential systems*, System Science and Mathematical Sciences **12** (1999), 43–60. 2
- [Mis93] Bhubaneswar Mishra, *Algorithmic algebra*, Texts and Monographs in Computer Science, Springer-Verlag, New York, 1993. MR MR1239443 (94j:68127) 5
- [Riq10] F. Riquier, *Les systèmes d'équations aux dérivées partielles*, 1910. 1
- [Rit50] Joseph F. Ritt, *Differential Algebra*, American Mathematical Society Colloquium Publications, Vol. XXXIII, American Mathematical Society, New York, N. Y., 1950. MR MR0035763 (12,7c) 1, 16
- [Ros59] Azriel Rosenfeld, *Specializations in differential algebra*, Trans. Amer. Math. Soc. **90** (1959), 394–407. MR MR0107642 (21 #6367) 13
- [Sei10] Werner M. Seiler, *Involution*, Algorithms and Computation in Mathematics, vol. 24, Springer-Verlag, Berlin, 2010, The formal theory of differential equations and its applications in computer algebra. MR MR2573958 2, 10
- [sGH09] Xiao shan Gao and Zhenyu Huang, *Efficient characteristic set algorithms for equation solving in finite fields and application in analysis of stream ciphers*, Cryptology ePrint Archive, Report 2009/637, 2009, <http://eprint.iacr.org/>. 1
- [Tho37] Joseph M. Thomas, *Differential systems*, AMS Colloquium Publications vol XXI, 1937. 1, 3, 16
- [Tho62] ———, *Systems and roots*, The William Byrd Press, INC, Richmond Virginia, 1962. 1, 3, 16
- [Wan98] Dongming Wang, *Decomposing polynomial systems into simple systems*, J. Symbolic Comput. **25** (1998), no. 3, 295–314. MR MR1615318 (99d:68130) 2, 3
- [Wan01] ———, *Elimination methods*, Texts and Monographs in Symbolic Computation, Springer-Verlag, Vienna, 2001. MR MR1826878 (2002i:13040) 2
- [Wan03] ———, *epsilon: description, help pages and examples of use*, 2003, (<http://www-spiral.lip6.fr/~wang/epsilon/>). 2, 16
- [Wan04] ———, *Elimination practice*, Imperial College Press, London, 2004, Software tools and applications, With 1 CD-ROM (UNIX/LINUX, Windows). MR MR2050441 (2005a:68001) 2
- [Wu00] Wen-Tsun Wu, *Mathematics mechanization*, Mathematics and its Applications, vol. 489, Kluwer Academic Publishers Group, Dordrecht, 2000, Mechanical geometry theorem-proving, mechanical geometry problem-solving and polynomial equations-solving. MR MR1834540 (2003a:01005) 1
- [Yap00] Chee K. Yap, *Fundamental problems of algorithmic algebra*, Oxford University Press, New York, 2000. MR MR1740761 (2000m:12014) 5